REMARKS

In the Office Action, the Examiner objected to claims 1-4, 6, 11, 13, 14, and 16-19, rejected claims 7-10 and 22-25 under 35 U.S.C. 112, and rejected claims 1-6, 11, 13-21, and 26 under 35 U.S.C. 102(e) as being unpatentable over Hummel (U.S. Patent No. 6,333,918). In this Amendment, Applicants have not canceled or added any claims. Accordingly, claims 1-11 and 13-26 will be pending after entry of this Amendment.

I. Amendments to the Specification

The specification has been amended to remove any hyperlinks and/or other form of browser-executable code from the specification.

II. Claim Objections

In the Office Action, the Examiner objected to claims 1-4, 6, 11, 13, 14, and 16-19 for informalities. Specifically, the Examiner stated that the terms "encoded sub-networks" and "local functions" used in these claims were not clear. The Applicants have amended the claims to address the Examiner's objections thereof. In particular, claims 1 and 13 now recite a method for "specifying sub-networks." Also, claims 1-4, 6, 11, 13, 14, and 16-19 have been amended to contain the term "node function" instead of "local function," a node function being a function of a node of a graph.

III. Rejections under 35 U.S.C. 112, Second Paragraph

In the Office Action, the Examiner rejected claims 7-10 and 22-25 under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which Applicant regards as the invention. Specifically, in these claims, the Examiner found insufficient antecedent basis for the element of "particular threshold complexity" in defining graphs. The Applicants has amended claims 7 and 22 to address the

Examiner's rejections thereof. Specifically, claims 7 and 22 have been amended to include the phrase "each graph has a complexity that relates to at least one structural attribute of the graph."

IV. Rejections under 35 U.S.C. 102(b)

In the Office Action, the Examiner rejected claims 1-6, 11, 13-21, and 26 under 35 U.S.C. 102(e) as being unpatentable over Hummel (U.S. Patent No. 6,333,918, hereinafter Hummel). The Applicants respectfully traverse. Amended claim 1 recites a method of specifying sub-networks comprising:

> a) defining a plurality of graphs, wherein each graph has a set of nodes;
>
> b) for each graph of the plurality of graphs, specifying two or more different sets of node functions, wherein each set of node functions for the graph includes one node function for each node of the graph, and the combination of the graph with one of the sets of node functions specified for the graph specifies a sub-network;
>
> c) for each graph of the plurality of graphs, storing the graph and the sets of node functions specified for the graph; and
>
> d) for each particular specified sub-network, storing an identifier that specifies the set of particular node functions and the particular graph that specify the particular sub-network.

Hummel discloses a method for determining a route from a source node to a destination node in a communications network and does not relate to a method of specifying sub-networks. In particular, Hummel does not disclose, teach, or even suggest each limitation of claim 1.

For example, Hummel does not teach or suggest specifying, for each graph of the plurality of graphs, **two or more different** sets of node functions for the graph where each set of

node functions includes one node function for each node of the graph, as required in claim 1. In fact, Hummel does not even discuss specifying a set of functions for a graph. Solely for the sake of argument, even if it is assumed that Hummel discloses **one** set of functions for a graph, no where in Hummel is it taught or suggested that **two or more different** sets of functions for a graph are specified, as required in claim 1. Applicants respectfully request that the Examiner cite the specific portion(s) of Hummel that disclose this feature of claim 1.

In addition, Hummel does not teach or suggest storing an identifier that specifies a set of functions and a graph that specify a sub-network, as recited in claim 1. In the route determination method, Hummel discloses the use of a pointer. However, as defined in Hummel, a pointer points to a node link pair that contains either the received physical node which is the lowest in the hierarchy or else one of its ancestor nodes (*see* e.g., col. 5, line 65 to col. 6, line 2 and col. 8, lines 23-25, and col. 9, lines 37-42). As such, the pointer used in Hummel does not specify a set of functions and a graph that specify a sub-network, as required in claim 1. Applicants respectfully request that the Examiner cite the specific portion(s) of Hummel that disclose this feature of claim 1.

As such, Applicants submit that claim 1 is in allowable form. Claims 2-11 are dependent on claim 1 and are allowable for at least the same reasons as claim 1.

Independent claim 13 recites a method of specifying sub-networks comprising:

a) specifying a graph with a set of nodes;

b) storing the graph;

c) storing first and second sets of node functions, wherein each set includes a node function for each node of the graph, wherein the combination of the graph and the first set of node functions specifies a first

sub-network, and the combination of the graph and the second set of node functions specifies a second sub-network;

d) for the first sub-network, storing a first identifier that specifies the graph and the first set of node functions; and

e) for the second sub-network, storing a second identifier that specifies the graph and the second set of node functions.

Hummel does not teach or suggest each recited feature of claim 13. For example, Hummel does not teach or suggest storing first and second sets of functions where each set includes a function for each node of a graph. Similar to the discussion above in relation to claim 1, even if it is assumed that Hummel discloses **one** set of functions for a graph, no where in Hummel is it taught or suggested that **two** sets of functions for a graph are stored, as required in claim 13. Also, Hummel does not teach or suggest storing an identifier that specifies a set of functions and a graph for a sub-network, as recited in claim 13. The reasons for this are discussed above in relation to claim 1.

As such, Applicants submit that claim 13 is in allowable form. Claims 14 and 15 are dependent on claim 13 and are allowable for at least the same reasons as claim 13.

Claims 16-26 are computer readable medium claims that are similar to claims 1-11. Accordingly, Applicants submit that these claims are patentable over the cited art for the same reasons as stated above for claims 1-11.

CONCLUSION

Based on the foregoing remarks, Applicants believe that the rejections and objections in the Office Action of October 29, 2004 are fully overcome and that the application is in condition for allowance. If the Examiner has any questions regarding the case, the Examiner is invited to contact Applicants' undersigned representative at the number given below.

Respectfully submitted,

STATTLER, JOHANSEN & ADELI LLP

Dated: 1/31/05

Gregory Suh
Reg. No. 48,187

Stattler Johansen & Adeli LLP
PO Box 51860
Palo Alto, CA 94303-0728
Phone: (650) 752-0990 ext.104
Fax: (650) 752-0995

Standard Cell Development Flow

Jos Dreesen

Philips Components Nijmegen

Abstract

This paper decribes the new standard-cell development flow, being used at Philips Components Nijmegen. The new flow will integrate all aspects of developing a standard-cell library into one program, resulting in a dramatic decrease in development effort and providing the user with a quick release of an errorfree library. Quality is maintained by virtue of a built-in quality control tool. A stick-editor will reduce layout efforts to a few minutes per cell.

1) Past methods

Methods used up to now mainly consisted of using stand-alone programs for constructing the layout (e.g. some sort of polygon- pusher program), generating models, extracting timing data (e.g. SPICE) and documenting the library (mostly a DTP-type of program). The different views of a cell (layout, timing & behaviour) were not correlated to each other, which resulted in error-prone libraries. Manual verification revealed most of these errors, but not all. The development effort was immense : several man-years for a library was commonplace. Worst of all, changes in processes or parameter-sets often meant redoing major parts of this work, resulting in error-prone and very costly maintenance.

2) New method

To improve on these past methods, we are setting up a new flow, which will integrate all aspects of library-development into one program. We will list these aspects in some more detail and comment on the differences with previous approaches to the problem.

2.1) Layout

Layout for most of the core-cells is done with the use of an in-house developed symbolic editor called STICK. Stick displays a coarse grid on which all basic parts of a cell can be placed. These include transistors, contacts and wiring. Only those parts that directly describe the functionality of the cell are entered. This symbolic layout (also called stick diagram) is virtually process-independant. Another program, named CELL, is used to flesh out the stick-diagram into an actual layout. Backgate contacts, power supply tracks and wells will be added automatically, resulting in a complete layout of a fully functional cell. The process-information that CELL needs is gathered from the process-description file used in the GDT layout-tools, GDT being an IC-design environment from Silicon Compiler Systems. GDT is used troughout the flow as the basic tool concerning the cell's layout. These layouts are given in the L-language, which is a C-like language especially defined by GDT for the purpose of describing layouts. This method of using symbolic layout retains the flexibility of hand-layout, while greatly speeding up the

TH0316-0/90/0000/0450/$01.00 © 1990 IEEE

design-process. However, this approach is ill suited for making IO-cells. These are made using the GDT-Led tool or the PHILIPS-CMSK tool. Our IO-cells are 'mask programmable' i.e. all IO-cells are variants of one basic cell, each cell being programmed for the desired function by varying the first- and second layer metal only.
When compared with previously used methods, which consisted of manually composing each and every cell with a polygon- pusher, the new method greatly improves the designers productivity and reduces the chance of layout-errors.

2.2) Models

Simulation models are made with the Mentor Graphics Neted tool. They consist of gate-level schematics, which combined with the primitive models for the basic gates result in a model that can be used in the Mentor Graphics Quicksim simulator. A symbol for each cell will also be made. The approach used here is not that much different from the old one, the major difference being that the Neted tool has been neatly integrated into the design- flow program. Simulation models for SIMON, a Philips logic level simulator, are no longer provided which means that we have only one basic simulator. Schematics must be entered only once and will remain valid for all libraries to come, unless one decides to alter the way a cell is built up (for instance, replacing a transmission gate exor by a fully gated exor) . Note that the simulation models do not contain any timing data, which means that they are process-independent. As will be explained later on, all timing data will be put in one file from which the models can fetch the relevant data at run-time. Different libraries can therefore use the same models, but with a different timing file for each process or library. The schematics constructed in this stage will also be used in the documentation generator.

2.3) Timing data

The timing data for the cells, which include delay-figures, setup-and hold timing, minimal pulse-width and so on are calculated by simulating their transistor-level behaviour with the use of a low-level circuit simulator (e.g. SPICE) This task, also called 'characterisation of a library', involved laboriously setting up each spice run and interpreting the result-file. Timing data were manually included in the models. It goes without saying that this method was extremely error-prone and very time- consuming : six months were needed to extract all timing data for a a medium-sized library (+/- 130 cells). To get rid of this major hurdle, we constructed a program called CELLCHAR which is capable of generating spice-input decks, running spice and interpreting the results to obtain all required timing figures. It is steered by attaching a type-label to each cell. This label is the only human input required for the whole characterisation process. As these labels are fairly simple (e.g. NAND, NOR , JK_FF, LATCH etc.) there is little room for errors. Cellchar can then set up the correct spice jobs by reading the cell's pin-names and applying the right waveforms to the right inputs. After running the spice-simulations, all extracted data are written into the so-called widgets control file, from which the models can read their timing behaviour at run time. Currently cellchar implements the following measurements :

- o Determination of fanin and fanout.
- o Determination of switching levels.
- o Determination of intrinsic and load-dependant delays.
- o For tri-state cells : cut-off time.
- o For flip-flops : setup & hold times, removal times and minimal puls-widths.

Yet to be implemented are current consumption figures. The switching levels mentioned here are the voltages between which the delay-figures will be measured. They are calculated in such a way as to make sure that the delay- figures are as independent of the input-slopes as possible. Another task of cellchar is to split up the delay-figures that were found : the calculated delays are path-delays i.e. they give the delay between one input and one output. Most models, however, require that the delays are assigned to one in- or output pin only. This translation from path- to pin-delays will be done automatically by the cellchar program. The characterisation still involves massive amounts of spice-runs and hence a great deal of CPU-time (130 cells : one week cpu-time on a 1-MIPS machine), but since human intervention has been cut to an absolute minimum the whole process is speeded up by at least a factor 20. The current approach makes it possible to characterize a library anew for different simulation conditions, whereas in the past we had to be content with derating figures only. The automatic nature of the current characterisation approach makes errors virtually impossible and results in more reliable timing models.

2.4) Behaviour specification

A new element is the behaviour specification in the form of a truth-table. Previously, truth tables were only used as a somewhat neglected part of the documentation. In the new flow, the truth table is 'the reference' for the logic behaviour of the cell. It is the standard against which both models and layout will be measured during the quality control procedure. And of course, these truth tables will form a major part of the documentation. These truth-tables are entered manually. Since the tables are totally process- and library-independent they are identical for each library and hence the somewhat elaborate entering of these tables has to be carried out only once.
A sample truth-table for a flip-flop is :

```
(   d    cl    sen    re   |   q    qn    )
{
   X     X     H      L    |   L    H  ;
   X     X     L      H    |   H    L  ;
   X     X     L      L    |   L    L  ;
   L     R     H      H    |   L    H  ;
   H     R     H      H    |   H    L  ;
   X    ~R     H      H    |   Q   ~Q  ;
}
```

2.5) Documentation

Documentation is made with the Mentor Graphics DOC and PICED tools. A script has been written that fills a predefined sheet with the desired information. This script will find schematics, layout, timing figures and truth tables and fit all these nicely onto two or more pages. Binding text is entered manually. Although not as dramatic as with layout and characterisation, it can be stated that generation of the documentation has been speeded up considerably.

3) The flow

The different parts of the development flow are strongly interrelated. In this section we will describe the various interrelationships. The checking procedures will be discussed separately in the next chapter. The flow-diagram on page 6 will guide you through the different parts of the flow : dashed boxes represent the basic input to a library, slanted ones are the output from the flow and all other boxes represent intermediate steps.
On the flow-diagram you can see the three main inputs for a new ASIC-library : Layout, models and a behavioural description.
The layout will, in most cases, be given in the GDT L-language. (Remember that

symbolic layouts were expanded into exactly that format) However, interfaces are provided enabling the user to import layout in foreign formats such as GDSII or VLSI-format. Cells that are made with the stick-editor already have the topological data that is required to generate bounding boxes for different auto-routers supported in the flow. All other cells require the user to provide the system with a topology-list. Currently bounding boxes are generated for the GDT-Autocells and the VLSI-router. After this input has been given, the program will continue by converting the layout for each cell to a spice-level transistor-netlist (using the LOCAL circuit extraction program) and to a switch-level netlist (using the GDT Led tool) The transistor-level netlist will be used as a base for the characterisation process (CELLCHAR program) and as a reference for the netlist-check and timing-check which are carried out later on. The widgets control file coming out of the CELLCHAR program and which contains all timing information, is one of the endproducts of the design flow. It will be used twice inside the remaining flow : once in the documentation and once in the timing check phase.
The models (schematics) entered in the Neted tool are used all over the design flow : most importantly, they are the basic simulation model. Models for the other simulators (currently the GDT LSIM and the MACH simulator) will be created by automatic translation of these Neted models. Furthermore, the basic gates used in these models will be substituted by their transistor-models which results in a transistor-level netlist for the whole model. This is done both for the QUICKSIM-model (= Neted model) and the LSIM-model. These netlists will be used in the netlist-checking procedure.
-The behaviour description or truth-table is used mainly as a reference for the checks. For this purpose, the tables will be translated into input-decks for the different high-level simulators. They will also be used in the document generator.

4) Quality control

Quality is maintained throughout the development flow by mandatory control steps. Currently four of them are prescribed : the DRC-check, functional check, netlist check and timing check. The first check is a design rule check : this is done with the industry standard DRACULA design rule checker. Delivering the DRC-job and interpreting the result-file is done automatically. The scope of this check goes beyond the normal design-rule checks : border-cells will be added to the left and right of each cell to ensure that no design- rule violation can occur on any possible arrangement of cells. Also feed-through locations (locations where the autorouter can route through the cell) are checked to ensure that a wire can indeed be passed through them. The DRC-check will produce a 'DRC ok' flag indicating that the cell is correct or it will produce a list of errors for each erroneous cell. This test assures that the cell's layout adheres to all process-releated rules.
Note that this does not mean that the cell will behave as planned. To ensure this, another test will be carried out : the functional (or behaviour) check. This check will compare the behaviour from the switch-level netlist, which closely represents the layout, against the truth-table. To this end, a program will read the truth-tables and translate them into input-files for the Lsim simulator. Don't care states will be expanded into one H and one L state, sequential cells will undergo the proper set/reset actions before being triggered so as to ensure a correct evaluation of the entire truth-table. The program then invokes this simulator and will compare the simulator's output with the expected output values, as given in the truth table. A flag indicating 'layout adheres to the truth- table' will be given. It assures that the layout does function as described in the truth-table or, in other words, the cell does function as planned.

The second part of this check repeats these steps with the quicksim- models and the models derived from these : they will also be cycled through the entire truth-table and once again the results from the simulations will be matched against the values from the truth-table. If they match, it has been proven that the models do adhere to the truth table. A flag indicating this fact will be given. If both layout and models match the truth-table, it is implied that the models match the layout and can be used as a representation of the cell. Note that all simulations in this check are carried out without regard to timing : we are dealing with the logic behaviour only.

The third check is a netlist-comparision: the transistor-netlist that was produced by the LOCAL program and which is a mirror image of the actual layout is compared with the netlists produced by expanding the gate-level models. This expanding involves, as explained earlier, the substitution of each gate from a gate-level model by its transistor- netlist. The comparator-program (GIMINI, a recent PHILIPS program) will match the transistors in the netlist and will produce flags that state whether or not all netlists are identical. If it is proven that the circuitry of both layout and expanded models are identical, the way is open to apply layout-versus-schematic checks on ASIC's made with this library.

The last check to be applied is a timing check. The simulations as done in the functional check will be repeated, but this time the widgets control file, containing the timing data, will be taken into account. Once more the cell will be cycled through the truth-table, using the high level Quicksim and Lsim simulators. As a reference the same stimuli will be used to drive a SPICE transistor- level simulation. The delay-times resulting from the simulations will be compared to see how close they match. Due to the fact that we are comparing two different kinds of simulations, there will always be a difference between them. However, we can check if the following conditions are true : delay-times of models must be equal or greater than delay-times from the layout, but not be more than 10 % greater. If these conditions are fulfilled, we can be assured that the figures as given in the widgets-control file will, in conjunction with the high-levels models, yield a good representation of the total behaviour of a cell. If not, the process of translating path-to pin-delays was carried out poorly. If all four checks are completed the different views of the cell (layout, model, timing & documentation) are released for public use. That is, if and only if all four check-flags were set 'ok'.

5) Conclusion

The new standardcell design flow as presented here, will result in a more reliable, less error-prone and much quicker development and release of libraries. Maintaining these libraries is much easier due to the consistent way in which the libraries have been created. The generation of timing-figures and their use in models and documentation is entirely automated and therefore error-free.

Documentation is made using the same data as used for the construction of the cells and is therefore consistent with these cells. Portability to other processes is greatly enhanced because all basic inputs to the flow do not contain process- dependent information. Use of symbolic layout insures that even layout is easily transferred to a new process. Response on user questions and demands (cell on demand procedure) is quicker and will be of the same high standard as the library itself. Use of the flow also means that the library will be available in a number of formats : currently these include GDT-standardcells, VLSI-standardcells and Mentor Graphics models with GDSII files. Finally , all cells have undergone the built-in rigid quality-check, ensuring their correctness.